

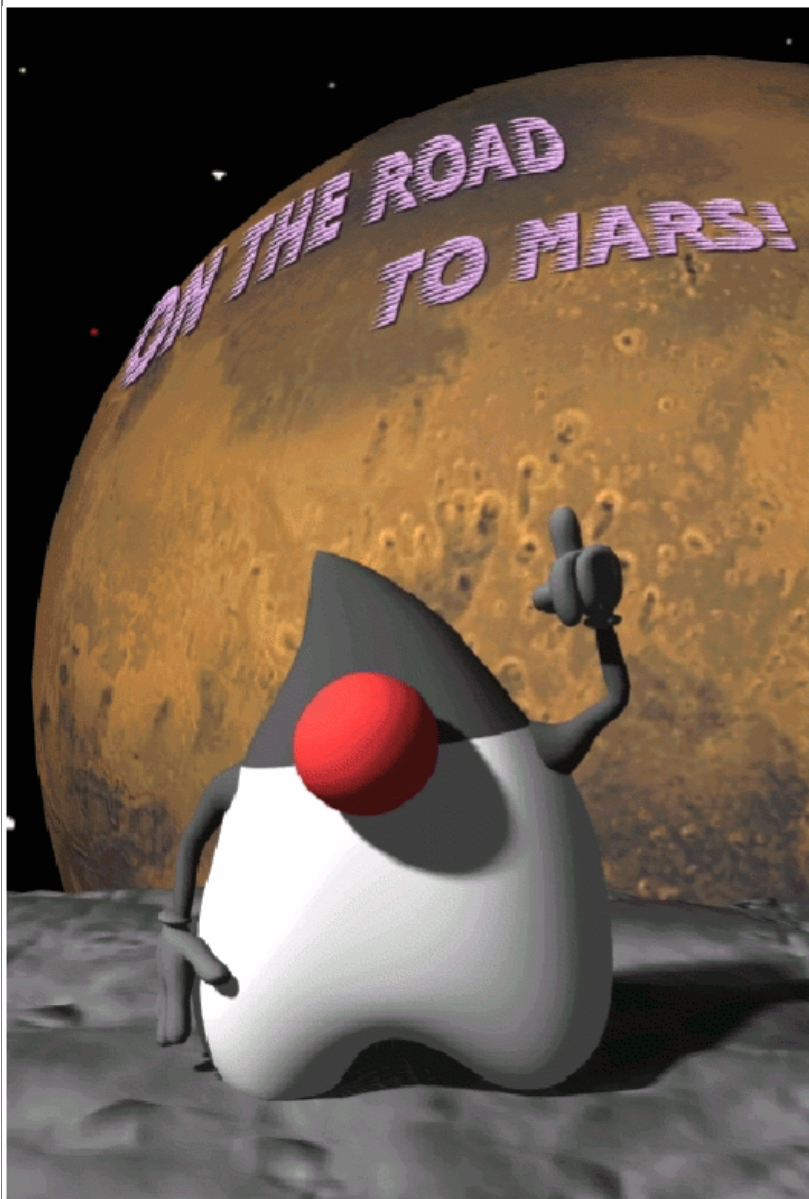


Golden Gate Project: RTSJ on a Mars Rover

Brian Giovannoni
Carnegie Mellon University

Real-Time Java for
Mission and Safety Critical Applications



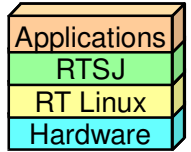


Project Goldengate

Sun Labs, NASA and Carnegie Mellon University send Java™ to Mars

Sun Labs, NASA JPL (Jet Propulsion Laboratory) and Carnegie Mellon University scientists are working together on a project to implement JPL's Mission Data System software architecture using the Real-Time Specification for Java development platform. A MDS implementation in RTSJ is under consideration for the Mars Smart Lander mission, planned to launch in 2009

Physical devices(vehicles, spacecraft, manufacturing equipment, etc) are currently being designed with the assumption that software will play a much larger part of the device's control aspect. Traditional design of these devices has relied primarily on analog electrical, hydraulic, and pneumatic control structures but various trends have caused current designs to depend more on software for control. Many organizations have begun, with the help of the software community, to create software architectures for these next generation devices and are looking for the most advanced software development platforms capable of supporting the rigorous physical, temporal, and complexity demands for device control.

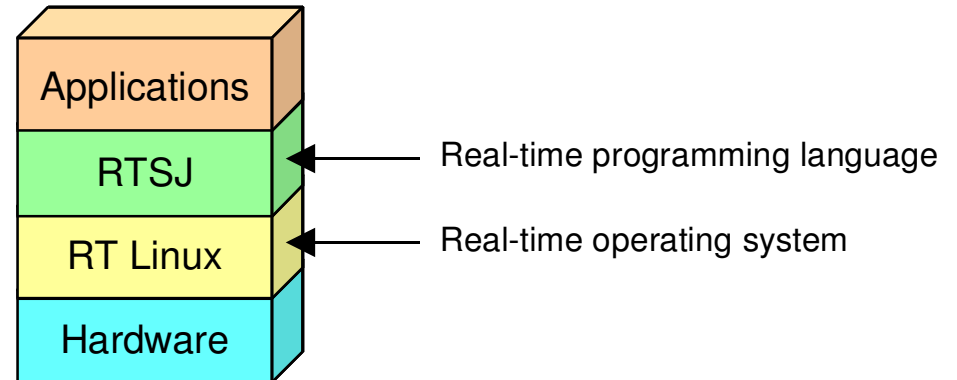


Project Overview

Approach

- Evaluate real-time Java against FSW demands
 - Target challenges cited in Nov. 1999 Nichols report "Using Java for Flight Implementation"
- Use MDS framework/code as a test bench
 - Same rover functionality as MDS/C++/VxWorks
 - Side-by-side performance comparison
- Leverage Distinguished Visiting Scientists
 - Dr. James Gosling, creator of Java, Sun Microsystems
 - Dr. Greg Bollella, lead of RTSJ, Sun Microsystems

A Modern Software Platform for Real-Time Embedded Systems



(RTSJ = Real-Time Specification for Java)

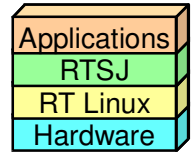
Work Plan

- FY 2003:
 - Install TimeSys Linux and RTSJ JVM on Rocky7
 - Prototype, run, and measure major MDS components
 - Run AFRL/Boeing test suite for RTSJ
- FY 2004:
 - Extend to run full MSL test scenario on Rocky7
 - Compare performance to C++/VxWorks version
 - Complete evaluations on latest product release

Evaluation Criteria

- Performance measurements
 - CPU usage
 - Throughput
 - Real-time response & timing jitter
 - Cache hit ratio
- Maturity of RTSJ and Linux/RT technology
- Multi-language development
- Application development effort

Overview of Mission Data System



Problem Domain

Autonomous control of physical systems

- Developed for unmanned space science missions involving spacecraft, landers, rovers, and ground systems
- Broadly applicable to mobile and immobile robots that operate autonomously to achieve goals specified by humans
- Architecturally suited for complex systems where “everything affects everything”



Approach

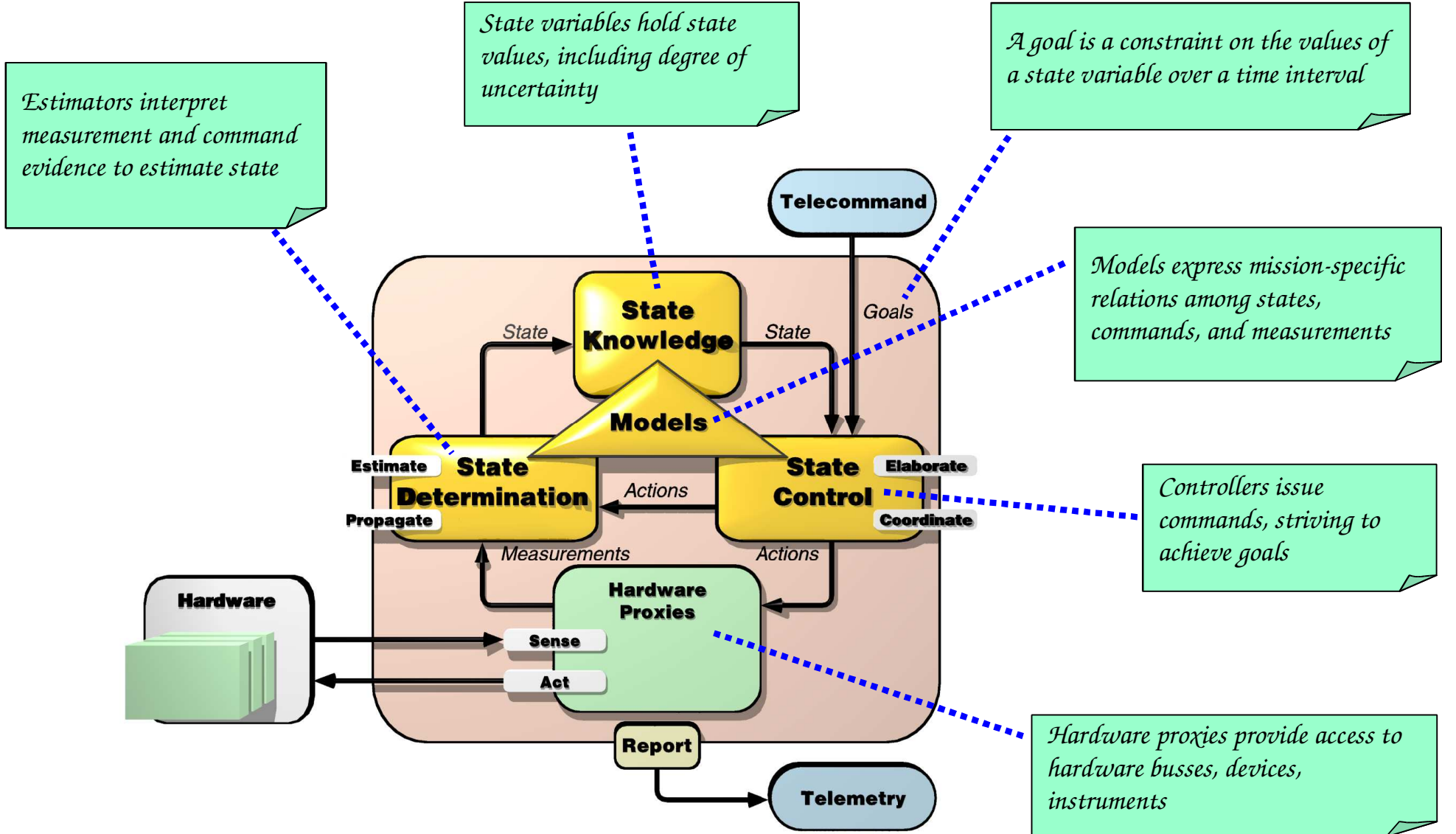
Product line practice to exploit commonalities:

- Define a reference architecture to which missions/products conform
- Provide framework software to be used and adapted
- Define processes for systems engineering and iterative software development

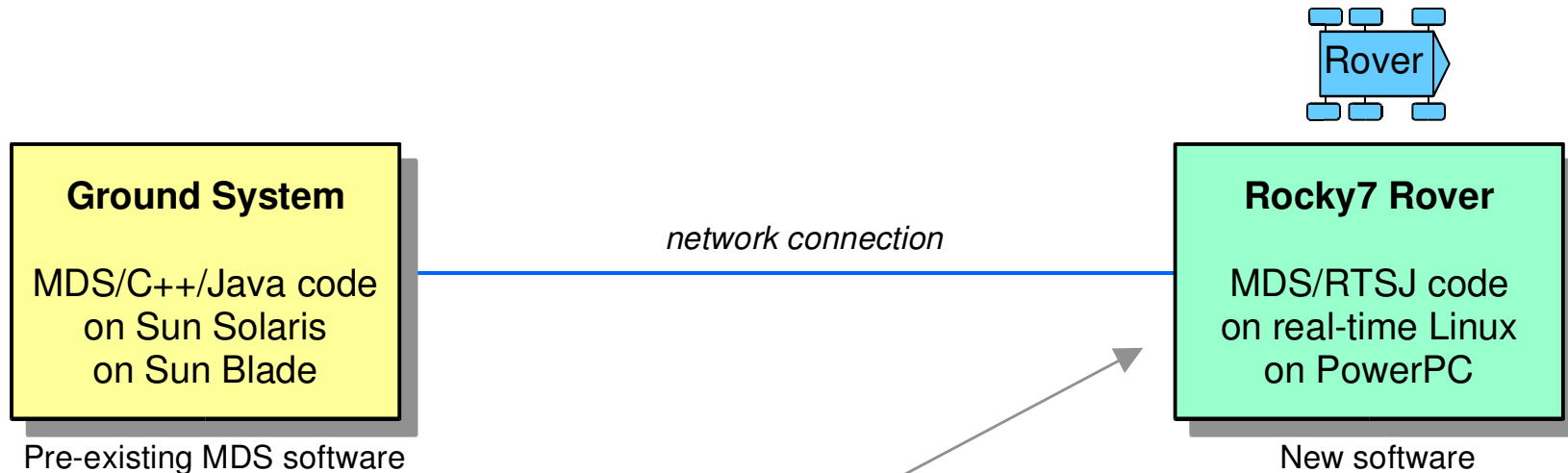
MDS Products

- Unified flight, ground and test architecture
- Orderly systems engineering methodology
- Frameworks (C++ and Java)
- Processes, tools, and documentation
- Examples
- Reusable software

MDS State-Based Architecture



Accomplishments to Date (1)



Observation:

Our rover software is exercising RTSJ features that enable hard real-time computation:

- *no-heap real-time threads*
- *scoped memory*
- *immortal memory*
- *asynchronous event handler*
- *periodic parameters*
- *one-shot timer*

- TimeSys Linux installed
- TimeSys JTime virtual machine installed
- Linux driver for digital I/O board working
- LM629 hardware adapter working
- Driving motor controller working
- Executable goals working
- Downlink data transport working

Current results:

- We are spinning wheels on Rocky7!
- And we see telemetry on Ground!
- Using RTSJ version of MDS frameworks!

Accomplishments to Date (2)

- *We have converted a substantial body of MDS/MSL software from C++ to RTSJ and run it on Rocky7*

Amount of software in repository :

- *31 packages*
- *343 classes (267 FW, 76 Test)*
- *2003 methods (1640 FW, 363 Test)*
- *12957 non-comment source statements*



Rocky 7

MDS Framework packages:

- *Components & connectors*
- *Data catalog*
- *Value history*
- *Data transport*
- *Math*
- *Physics*
- *Resources*
- *State knowledge*
- *Goal achievers*
- *Goal network*
- *Component scheduler*
- *Hardware adapter*
- *State types*
- *Utilities*

Rocky7 adaptation packages:

- *LM 629 device*
- *PCI device*
- *S720 device*
- *Motor*
- *Motor simulator*
- *Position & heading controller*
- *Angular position*

Accomplishments to Date (3)

· Documented RTSJ programming models for practitioners

- RTSJ introduces new memory areas that enable hard real-time programs to run without interference from the Java garbage collector
- The RTSJ rules for using these memory areas affect the programming model
- We documented a pattern of using scoped memory scratchpads and restricted memory pools as a good combination of Java style and reduced risk of programming error

· Through framework design, we have shielded some RTSJ complexity from adapters and facilitated testing

- Execution wrapper allows us to run functional tests of adaptation code on ordinary, non-RTSJ Java platforms (such as your desktop computer)
- Connectors shield the complexity of moving data between heap memory and non-heap memory

Accomplishments to Date (4)

·Outlined forthcoming performance metrics work

- Support comparisons among 3 platforms: RTSJ / Linux, C++ / Linux, C++ / VxWorks
- *Focused measures*: interrupt response latency, jitter, scheduling and dispatching overhead, garbage collection execution, and processor throughput, including numerically-intensive computation
- *System-level measures*: CPU utilization, cache hit ratio, memory footprint, and language-induced overheads, such as the necessity of moving data among RTSJ's different memory areas.

·RTSJ has had a big influence on new design for MDS/C++ component scheduler

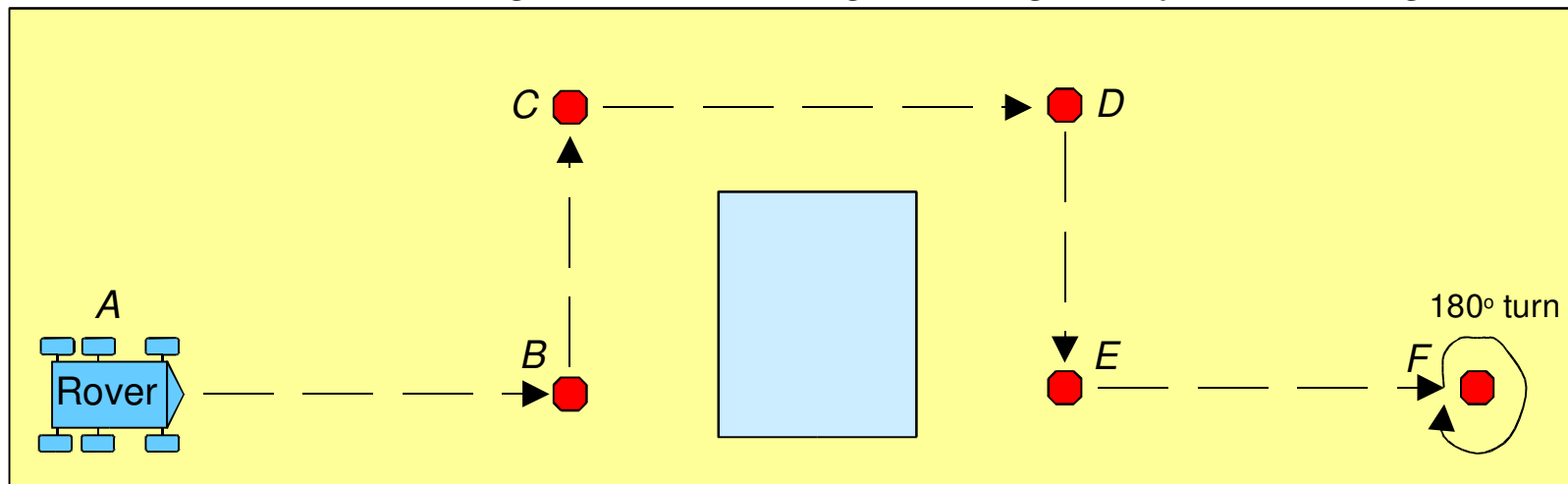
- RTSJ forces explicit, up-front consideration of execution model (and that's good)
- RTSJ features: RealtimeThread, NoHeapRealtimeThread, ReleaseParameters, AsyncEvent, ...

Deliverables Expected in FY 2003 (1)

- **Finish rover test scenario for JavaOne**

- Requires camera control (frame grabber driver, hardware adapter, controller, estimator)
- Requires coordinated 6-wheel driving & steering controller

Test Scenario for goal-driven driving, steering, and picture-taking



- *Rover follows the path A-B-C-D-E-F*
- *Rover takes still images periodically as it goes*
- *Traverse culminates in a 180-degree turn-in-place*

Deliverables Expected in FY 2003 (2)

- **RTSJ versions of MDS packages (a subset)**
 - Supports meaningful comparison between RTSJ/Linux and C++/VxWorks
 - Refactor as needed
- **Documented metrics for real-time platforms**
 - Establish data collection process
 - Run AFRL/Boeing RTSJ test suite on embedded PowerPC
 - Write initial tests and collect data
- **Initial written assessment of RTSJ/Linux platform with respect to risk factors highlighted in 1999 Nichols report:**
 - Real-time response
 - Performance
 - Multi-language development
 - Maturity
 - Application programming considerations (*our addition*)

Deliverables Expected in FY 2003 (3)

- **Expansion of programming model document for practitioners**
 - More on design tradeoffs, restrictions, pitfalls, useful idioms, best practices
 - Characterize application development complexity relative to C++
- **Feedback design suggestions to RTSJ Expert's Group**

Brian Giovannoni

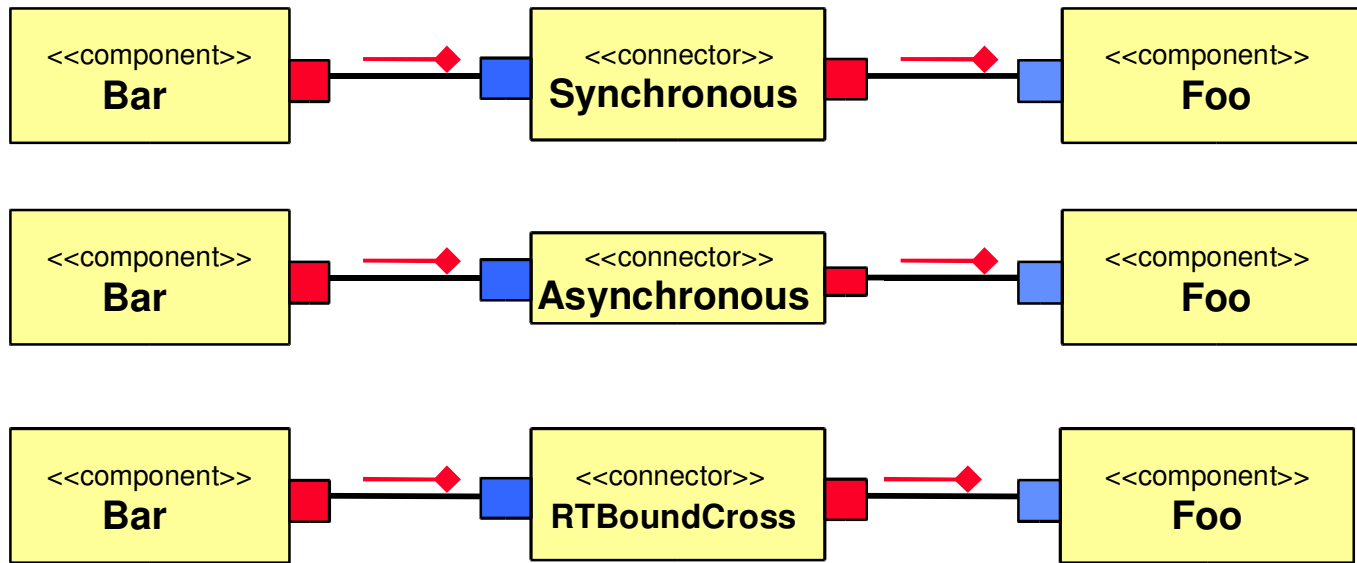
briang@west.cmu.edu

Reserves

Six Tracks of Work

- Main Track
 - development environment, integration & testing, demo scenario
- Component Track
 - Components and connectors designed for RTSJ
- Hardware Track
 - PPC boot, install Linux/RT, spin rover wheel on Dexter,
- Demo Track
 - Rocky7 rover scenario, setup, execution, logistics
- GDS Track
 - Transport data to ground, then display telemetry
- RTSJ Metrics Track
 - Jitter, throughput, CPU & memory usage, cache hit ratio, etc

Components & Connectors



- Goals

- Components are **reusable**
 - Component interaction logic separated from component functionality
 - Method calls from Bar to Foo are same regardless of connector type
- Components are **easy to write**
 - Adapter not concerned with communication details

Simple Code Example

```

public class Bar extends Component implements Executable {

    private MyInterface myInterface; ← Determined by Connector, set in architecture configuration

    public Bar() {
    }

    public void execute () {
        myInterface.baz ("Hello!");
        myInterface.faz (12);
    }

    public void init() {}

    public void setCallableInterface(Callable anInterface) {
        if ((myInterface.requiredInterface).isInstance(anInterface)) {
            myInterfaceObject = (MyInterface) anInterface;
        }
    }
}

```

Component functionality
 looks like a direct method invocation
 - no indirection for synchronous blocking calls

Instantiate all ports on component creation

■ Adapter written code ■ Potentially auto-generated